ESSLLI 1998 COURSE

Tableau-Based Theorem Proving

Reiner Hähnle

Department of Computer Science
University of Karlsruhe
D-76128 Karlsruhe
`http://i12www.ira.uka.de/~reiner`

June 30, 1998 – Draft version

Abstract of the Course

Tableau-based approaches to automated deduction have gained a lot of ground in recent years after many years during which they have been mainly considered for teaching and in proof theoretic studies. One reason for this development is that competitive implementations of first-order logic tableau provers are now available, but another reason for the renewed interest is the large number of applications for deduction in non-classical in linguistics, intelligent agents, knowledge representation to name just a few fields. I will present various tableau-related calculi in a uniform, modern framework with new, concise completeness proofs. Important implementations as well as basic techniques for accommodating non-classical aspects are discussed as well. A brief introduction into the required concepts of computational logic is included.

1. Introduction

Reasoning methods based on tableaux and their relatives gained a lot of attention in the past decade after a long period of near stagnation. One reason is that theoretical and implementational progress finally permitted to build tableau-based theorem provers [23, 42] that can compete [60] with state-of-the-art resolution-based systems. Another reason is the increased need for deduction in various non-classical logics for which tableau calculi are particularly well suited.

Today, a large number of refinements of tableau-like calculi aimed at efficient automated proof search are available. In fact, there are so many of them that it has become quite difficult for the non-specialist keep track of the main developments. The difficulty of this task is incrased by the plethora of names for closely related systems: connection tableaux, connection method, hypertableaux, matrices, matings, model elimination, model generation, near-Horn logic programming, SL-resolution all are relatives of each other.

In this paper I introduce the main lines of development of tableau-like calculi, as far as they are relevant for automated reasoning, in a uniform framework. At the same time I work out their mutual relationships and I classify the refinements according to various properties.

Most refinements of tableau calculi are defined and implemented only for clause normal form. Accordingly, after a brief treatment of tableaux for full first-order logic in Section 3, the bulk of the material is presented on the clause level. In Section 4 the main types of refinements of tableau-like calculi are defined and discussed. The history of tableaux-like proof methods is long and vined. Many key ideas were discovered several times independently. I trace the major developments in a very brief historical Section 5.

2. Preliminaries

Here some basic ingredients of computational logic are collected. This section cannot replace a proper introduction into logic and elementary issues of theorem proving. I recommend Fitting's book [18] as a background.

2.1. Syntax

A *first-order signature* $\Sigma = \langle P_\Sigma, F_\Sigma \rangle$ consists of a non-empty set P_Σ of predicate symbols and a set F_Σ of function symbols. For skolemization we do not use symbols from F_Σ but from a special infinite set F_{sko} of *Skolem function symbols* that is disjoint from F_Σ; the extended signature $\langle P_\Sigma, F_\Sigma \cup F_{\text{sko}} \rangle$ is denoted with Σ^*. The symbols in P_Σ, F_Σ and F_{sko} may be used with any non-negative arity. In addition, there is an infinite set Var of *object variables*.

Given a signature Σ, the sets $\mathcal{T}_\Sigma$ of *terms* and $\mathcal{A}_\Sigma$ of *atoms* over Σ are inductively defined by:

(i) Object variables and 0-ary function symbols from Σ are terms.

(ii) If $t_1, \ldots, t_n$ are terms, f is a n-ary function symbol from Σ, and p is a n-ary predicate symbol from Σ, then $f(t_1, \ldots, t_n)$ is a term and $p(t_1, \ldots, t_n)$ is an atom over Σ.

The *logical operators* are the connectives $\vee$ (disjunction), $\wedge$ (conjunction) and $\neg$ (negation), the quantifier symbols $\forall$ and $\exists$, and the constant operators *true* and *false*.

Given a signature Σ, the set $\mathcal{L}_\Sigma$ of *formulas*[1] over Σ is inductively defined by:

(i) *true*, *false* and atoms over Σ are formulas.

(ii) If ϕ is a formula, then $\neg\phi$ is a formula.

(iii) If $\phi_1, \ldots, \phi_n$, $n > 1$, are formulas none of which is a conjunction (disjunction), then $\phi_1 \wedge \ldots \wedge \phi_n$ (resp. $\phi_1 \vee \ldots \vee \phi_n$) is a formula.

(iv) If ϕ is a formula and $x \in \text{Var}$, then $(\forall x)\phi$ and $(\exists x)\phi$ are formulas. ϕ is called the *scope* of the quantifier.

Formulas that are identical up to associativity of $\vee$ and $\wedge$ are identified. A *literal* is an atom or a negated atom.

A *ground term (atom, literal, formula)* is a term (atom, literal, formula) that contains no object variables. The set of ground terms is abbreviated with $\mathcal{T}^0$. No difference is made between ground first-order formulas and propositional formulas.

The *complement* $\overline{\phi}$ of a formula ϕ is defined by: $\overline{\phi} = \psi$ if ϕ is of the form $\neg\psi$, and $\overline{\phi} = \neg\phi$ otherwise.

An occurrence of an object variable x in a formula is called *bound* if x occurs in the scope of a quantifier over x, it is called *free* otherwise. A formula without free variable occurrences is a *sentence*.

A *clause* is a sentence of the form $(\forall x_1) \cdots (\forall x_n) L_1 \vee \cdots \vee L_m$, where L_i are literals. For sake of readability the quantifier prefix of first-order clauses is usually

[1] Implication and equivalence are considered to be defined operators, i.e., $\phi \rightarrow \psi$ is the same as $\neg\phi \vee \psi$, and $\phi \leftrightarrow \psi$ is the same as $(\phi \wedge \psi) \vee (\neg\phi \vee \neg\psi)$.

not written (but assumed to be present). Note that clauses are particular formulas. When C, D are ground clauses $C \subseteq D$ means that every literal of C occurs also in D; $L \in D$ expresses that the literal L occurs in clause D. A clause is a *tautology* if it contains p and $\neg p$ for some atom p.

A *substitution* is a mapping $\sigma : \mathrm{Var} \to \mathcal{T}_\Sigma$. It is extended to terms and (sets of) formulas as usual:

- $\sigma(c) = c$, $\sigma(\mathit{true}) = \mathit{true}$, $\sigma(\mathit{false}) = \mathit{false}$
- $\sigma(s(t_1, \ldots, t_n)) = s(\sigma(t_1), \ldots, \sigma(t_n))$ for $s \in F_\Sigma \cup P_\Sigma$
- $\sigma(A \bullet B) = \sigma(A) \bullet \sigma(B)$ for $\bullet \in \{\neg, \wedge, \vee\}$
- $\sigma(QxA) = Qx\sigma'(A)$ for $Q \in \{\forall, \exists\}$, where $\sigma' = \sigma \backslash \{x/t | t \in \mathcal{T}\}$
- $\sigma(\{A_1, \ldots, A_n\}) = \{\sigma(A_1), \ldots, \sigma(A_n)\}$

If $\sigma(x) = x$ for all but finitely many $x \in \mathrm{Var}$ it is denoted $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$. Application of substitutions is usually written postfix (note that $\phi(\sigma \circ \rho) = (\phi\rho)\sigma = \phi\rho\sigma$). Without loss of generality we assume that the range of substitutions σ never contains object variables that occur bound in a formula ϕ to which σ is applied. When the t_i are ground terms one has a *ground substitution*. A *renaming* is a substitution ν which is a permutation such that the variables in its range are new in the context, where ν appears.

If T is a non-empty set of terms and $|T\sigma| = 1$, then σ is a *unifier* of T. It is a *most general unifier (MGU)* if for all unifiers ρ of T there is a substitution θ such that $\rho = \theta \circ \sigma$.

Proposition 2.1. *Unifiability of a finite set of terms can be decided and, when positive, an idempotent MGU computed in linear time.*

An *instance* of a clause $C = \forall x_1 \cdots \forall x_n L_1 \vee \cdots \vee L_m$ is an expression $(L_1 \vee \cdots \vee L_m)\sigma$, where σ is a renaming. When σ is a ground substitution one has a *ground instance* of C.

A substring ψ of a formula ϕ which is itself a formula is called a *subformula* of ϕ. If ψ is a subformula of ϕ and $\phi \neq \psi$ then ψ is a *proper subformula* of ϕ. If ψ is a proper subformula of ϕ such that there is no proper subformula ρ of ϕ with ϕ being a proper subformula of ρ, then ψ is an *immediate subformula* of ϕ.

As we deal with arbitrary formulas we have to account for the fact that a disjunctive subformula may occur negated and thus is implicitly a conjunctive formula etc. Also, the sign of a literal may be implicitly complemented.

Definition 2.1. *An occurrence of a subformula ρ of $\phi \in \mathcal{L}_\Sigma$ is*

(i) positive *if $\phi = \rho$,*

(ii) negative *(*positive*) if ϕ is of the form $\neg\psi$ and the occurrence of ρ is positive (negative) in ψ,*

(iii) positive *(*negative*) if ψ is an immediate subformula of ϕ, but $\phi \neq \neg\psi$, and the occurrence of ρ is positive (negative) in ψ.*

2.2. Semantics

Given a first-order signature Σ a *first-order structure* $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ consists of a non-empty set $\mathbf{D}$ called *domain* and an *interpretation* $\mathbf{I}$ that assigns to each $f \in F_\Sigma$ a mapping $\mathbf{I}(f) : \mathbf{D}^{\alpha(f)} \to \mathbf{D}$ and to each $p \in P_\Sigma$ a relation $\mathbf{I}(p) \subseteq 2^{\mathbf{D}^{\alpha(p)}}$.

A *variable assignment* is a mapping $\mu : \mathrm{Var} \to \mathbf{D}$. The *d-variant of μ at x* is

$$\mu_x^d(y) = \begin{cases} d & \text{if } x = y \\ \mu(y) & \text{otherwise} \end{cases}$$

In any first-order structure $\mathbf{M}$ a variable assignment μ is extended to terms:

$$\begin{aligned} x^{\mathbf{M},\mu} &= \mu(x) \text{ for } x \in \mathrm{Var} \\ f(t_1, \ldots, t_n)^{\mathbf{M},\mu} &= I(f)(t_1^{\mathbf{M},\mu}, \ldots, t_n^{\mathbf{M},\mu}) \text{ for } f(t_1, \ldots, t_n) \in \mathcal{T} \end{aligned}$$

Satisfiability of formulas ϕ in $\mathbf{M}$ and μ, written $(\mathbf{M}, \mu) \models \phi$ is defined as follows:

$(\mathbf{M}, \mu) \models \mathit{true}$	for all $\mathbf{M}$ and μ
$(\mathbf{M}, \mu) \models \mathit{false}$	for no $\mathbf{M}$ and μ
$(\mathbf{M}, \mu) \models p(t_1, \ldots, t_n)$	iff $(t_1^{\mathbf{M},\mu}, \ldots, t_n^{\mathbf{M},\mu}) \in \mathbf{I}(p)$ for $p(t_1, \ldots, t_n) \in \mathcal{A}$
$(\mathbf{M}, \mu) \models \neg\phi$	iff not $(\mathbf{M}, \mu) \models \phi$
$(\mathbf{M}, \mu) \models \phi_1 \wedge \cdots \wedge \phi_n$	iff $(\mathbf{M}, \mu) \models \phi_i$ for all $i \in \{1, \ldots, n\}$
$(\mathbf{M}, \mu) \models \phi_1 \vee \cdots \vee \phi_n$	iff $(\mathbf{M}, \mu) \models \phi_i$ for at least one $i \in \{1, \ldots, n\}$
$(\mathbf{M}, \mu) \models \forall x \phi$	iff $(\mathbf{M}, \mu_x^d) \models \phi$ for all $d \in \mathbf{D}$
$(\mathbf{M}, \mu) \models \exists x \phi$	iff $(\mathbf{M}, \mu_x^d) \models \phi$ for at least one $d \in \mathbf{D}$

A first-order Σ-structure $\mathbf{M}$ is a *model* of a set of formulas $M \subset \mathcal{L}_\Sigma$, denoted $\mathbf{M} \models_\Sigma M$, if $(\mathbf{M}, \mu) \models \phi$ for all $\phi \in M$ and variable assignment μ. ϕ is a *logical consequence* of M, denoted $M \models_\Sigma \phi$ if each model of M is also a model of ϕ. ϕ is *valid*, written $\models_\Sigma \phi$, when each Σ-structure $\mathbf{M}$ is a model of ϕ.

A first-order Σ-structure $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ is a *term structure* if $\mathbf{D} = \mathcal{T}^0$.

Proposition 2.2. *For all $\phi \in \mathcal{L}_\Sigma$ and sets of $\mathcal{L}_\Sigma$-sentences M: $M \models_\Sigma \phi$ iff $M \cup \{\neg\phi\}$ is unsatisfiable.*

Theorem 2.1. *If a sentence $\phi \in \mathcal{L}_\Sigma$ is satisfiable, then it has a Σ-term model.*

Assume f_Σ contains at least one constant, then $\mathcal{T}_\Sigma^0 \neq \emptyset$. A structure $\langle \mathcal{T}_\Sigma^0, \mathbf{I} \rangle$, where $\mathbf{I}(f)(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ for all $f \in F_\Sigma$, is called *Herbrand structure*.

Theorem 2.2 (Clausal Version of Herbrand's Theorem)**.** *Let S be a finite, unsatisfiable set of clauses. Then there is a finite set $\overline{S}$ of ground instances of S such that $\overline{S}$ is unsatisfiable.*

A proof is, for example, in [58].

3. The Tableau Method

3.1. Informal Introduction

It is common to view the tableau method as a proof by contradiction and case distinction (this view was already stressed by pioneers Beth [10] and Hintikka [30]). More precisely, it allows one to systematically generate subcases until elementary contradictions are reached. Let us go through a small example:

Assume we want to prove the following simple theorem from elementary set theory: for any sets P, Q, R, S, if (1) $S \cap Q = \emptyset$, (2) $P \subseteq Q \cup R$, (3) $P = \emptyset \rightarrow Q \neq \emptyset$, and (4) $Q \cup R \subseteq S$, then $P \cap R = \emptyset$.

We give a proof by contradiction, therefore, assume $P \cap R \neq \emptyset$. From (3) we know that there is an element $c \in P$ (case 1) or there is an element $d \in Q$ (case 2). We proceed with case 1. The assumption applied to c yields that $c \notin P$ (case 1.1) or $c \notin R$ (case 1.2). Case 1.1 is contradictory. In case 1.2 apply (2) to c: if $c \in P$ then $c \in Q$ or $c \in R$. But we know already that $c \in P$ and $c \notin R$ which leaves case 1.2.2, $c \in Q$, to consider. Using (1) similarly as the assumption yields case 1.2.2.1, $c \notin S$. Finally, from (4) we have: if $c \in Q \cup R$ then $c \in S$. Again, the second possibility is a contradiction, and case 1.2.2.1.1, $c \notin Q \cup R$, remains. Thus, $c \notin Q$, contradiction. The remaining case 2 is similar.

This proof is easier to follow if the case distinctions are displayed tree-like as in Figure 1. We observe that case distinctions could be generated schematically depending on the form of their premiss. At several points, premisses had to be suitably instantiated. (1), (2) and (4) are universally quantified, for instance, (1) says that *for all elements x, x cannot be both an element of S and of Q*. In automated theorem proving finding instances is done by *unification*—one tries to find a substitution that produces a contradiction in the current branch of the proof (in the example this is $\{x/c\}$). In general one needs, of course, to apply a premiss more than once during a proof. This *multiplicity* cannot be computed in advance (otherwise, first-order logic would be decidable). One of the problems that tableau methods must solve is to systematically enumerate "enough" (this is made precise later) instances of universally quantified formulas.

From premiss (3) one obtains existentially quantified expressions saying that at least one of P and Q must contain an element. In automated theorem proving such witness elements are produced by Skolemization: the existentially quantified variable is replaced by a "new" (again, this is made precise later) term.

3.2. Non-clausal Tableaux with Unification

3.2.1. Unifying Notation

The first step in formalizing the considerations in the previous section is to supply formal rules that tell in which way a formula is analyzed according to its leading connective. Smullyan [58] observed that some work can be saved if non-literal formulas are grouped into types which are treated identically: α for formulas of

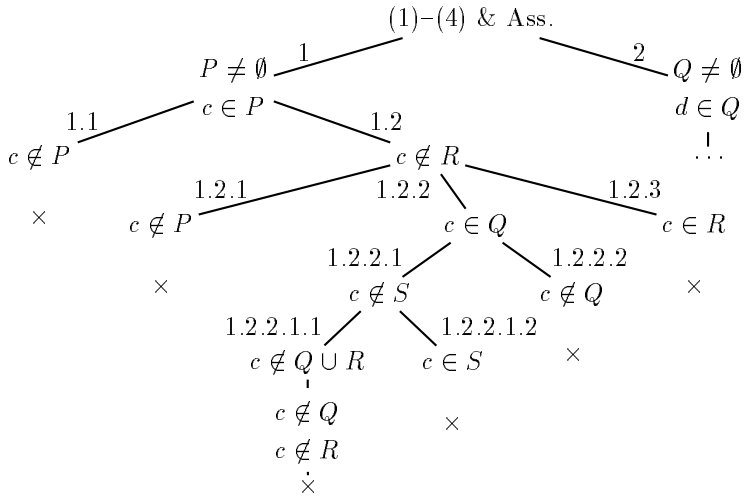


Figure 1. Structure of an informal proof by contradiction and case distinction.

conjunctive type, β for formulas of disjunctive type, γ for quantified formulas of universal, and δ for quantified formulas of existential type. Correspondence between formulas and their types is summarized in Table 1.

The letters α, β, γ, and δ are used to denote formulas of (and only of) the appropriate type. In the case of γ- and δ-formulas the variable x bound by the (topmost) quantifier is made explicit by writing $\gamma(x)$ and $\gamma_1(x)$ (resp. $\delta(x)$ and $\delta_1(x)$); accordingly $\gamma_1(t)$ denotes the result of replacing all occurrences of x in γ_1 by t. Associativity of $\wedge$ and $\vee$ justifies conjunctive and disjunctive formulas with an indefinite number of arguments.

Some authors [37, 58] prefer to work with *signed formulas*. These are expressions of the form $\mathrm{T}\,\phi$, $\mathrm{F}\,\phi$, where ϕ is a formula. Signed formula tableaux relate more directly to sequent calculi, because T-signed formulas play the rôle of formulas standing left of a sequent arrow while F-signed formulas are on the right (see also Section 4.7.1 below). In classical logic theorem proving there is no particular gain from signs, but in non-classical logics their use is indispensable [6, 24].

3.2.2. Tableau Rules

With unifying notation decomposition rules for arbitrary formulas can be given in a concise way.

In Table 2 expansion rule schemata for the various formula types are given. Premisses and conclusions are separated by a horizontal bar, while vertical bars in the conclusion denote different *extensions*. The formulas in an extension are implicitly conjunctively connected, and different extensions are implicitly disjunctively con-

α	$\alpha_1, \ldots, \alpha_n$
$\phi_1 \wedge \ldots \wedge \phi_n$	$\phi_1, \ldots, \phi_n$
$\neg(\phi_1 \vee \ldots \vee \phi_n)$	$\neg\phi_1, \ldots, \neg\phi_n$
$\neg\neg\phi$	ϕ
$\neg false$	$true$
$\neg true$	$false$

β	$\beta_1, \ldots, \beta_n$
$\phi_1 \vee \ldots \vee \phi_n$	$\phi_1, \ldots, \phi_n$
$\neg(\phi_1 \wedge \ldots \wedge \phi_n)$	$\neg\phi_1, \ldots, \neg\phi_n$

γ	γ_1
$(\forall x)(\phi(x))$	$\phi(x)$
$\neg(\exists x)(\phi(x))$	$\neg\phi(x)$

δ	δ_1
$\neg(\forall x)(\phi(x))$	$\neg\phi(x)$
$(\exists x)(\phi(x))$	$\phi(x)$

Table 1
Correspondence of formulas and their types.

nected. We use n-ary α- and β-rules, i.e., when the β-rule is applied to a formula $\psi = \phi_1 \vee \ldots \vee \phi_n$, then ψ is broken up into n subformulas (instead of splitting it into two formulas $\phi_1 \vee \ldots \vee \phi_r$ and $\phi_{r+1} \vee \ldots \vee \phi_n$, $0 < r < n$).

Type γ formulas are simply stripped from their quantifier while the quantifed variable is renamed into a object variable not ocurring elsewhere. Instantiation of free variables is delayed.

The δ-rule is the most technical rule. Its purpose is to replace an existential quantifier with a witness element or Skolem term. It incorporates two important optimizations: the first is that the choice of the witness element merely depends on the free variables in δ, not on all free variables on the current branch; secondly, the leading function symbol f of the Skolem term may be the same for δ-formulas which are identical up to variable renaming, formally:

Definition 3.1. *Given a signature $\Sigma = \langle P_\Sigma, F_\Sigma \rangle$, the function sko assigns to each $\delta \in \mathcal{L}_{\Sigma^*}$ a symbol $sko_\delta \in F_{\text{sko}}$ such that (a) $sko_\delta > f$ for all $f \in F_{\text{sko}}$ occurring in δ, where $>$ is an arbitrary but fixed ordering on F_{sko}, and (b) for all $\delta, \delta' \in \mathcal{L}_\Sigma$ the symbols sko_δ and sko'_δ are identical if and only if δ and δ' are identical up to variable renaming (including renaming of the bound variables).*

The purpose of condition (a) in the above definition of *sko* is to avoid cycles like: sko_δ occurs in δ' and sko'_δ occurs in δ.

Both improvements of the δ-rule together have the consequence that its conclusion can be computed locally to the formula δ—no "global" information is required.

$\dfrac{\alpha}{\begin{array}{l}\alpha_1\\ \vdots\\ \alpha_n\end{array}}$	$\dfrac{\beta}{\beta_1 \mid \cdots \mid \beta_n}$	$\dfrac{\gamma(x)}{\gamma_1(y)}$	$\dfrac{\delta(x)}{\delta_1(sko_\delta(x_1,\ldots,x_n))}$
		$y \in \mathrm{Var}$ is new to the tableau.	$x_1,\ldots,x_n$ are the free variables in δ.

Table 2
Rule schemata for tableaux with unification.

3.2.3. Tableau Proofs
As was hinted at already, tableau proofs are trees whose nodes are formulas that are (sub)goals in the proof and the tree structure gives the logical dependence between them. Assume we want to prove that a set of sentences Φ logically implies a sentence ψ. By Proposition 2.2 this amounts to checking that the set of sentences $\Phi \cup \{\neg\psi\}$ is unsatisfable.

Definition 3.2. *Let Σ be a first-order signature. A* tableau *(over Σ) is a finitely branching tree whose nodes are formulas from $\mathcal{L}_{\Sigma^*}$. A* branch *in a tableau T is a maximal path in T.*[2] *Given a set Φ of sentences from $\mathcal{L}_\Sigma$, a* tableau for Φ *is constructed by a (possibly infinite) sequence of applications of the following rules:*

(i) *The tree consisting of a single node true is a tableau for Φ (initialization rule).*

(ii) *Let T be a tableau for Φ, B a branch of T, and ψ a formula in $B \cup \Phi$. If the tree T' is constructed by extending B by as many new linear subtrees as an instance of a tableau rule schema in Table 2 with premiss ψ has extensions, and the nodes of the new subtrees are the formulas in the extensions of the rule instance, then T' is a tableau for Φ (expansion rule).*

(iii) *Let T be a tableau for Φ, B a branch of T, and ψ and ψ' literals in $B \cup \Phi$. If ψ and $\overline{\psi'}$ are unifiable with MGU σ, and T' is constructed by applying σ to all formulas in T (i.e., $T' = T\sigma$), then T' is a tableau for Φ (closure rule).*

The last item of the previous definition incorporates two simplifications: first, MGUs are used instead of arbitrary substitutions; second, ψ and ψ' are literals, not arbitrary formulas. The former is crucial, because there are only finitely many MGUs of (complemented) formulas on a finite tableau. If Φ is finite this implies that there are systematic procedures for enumerating the (finite) tableaux for Φ.

The branches in a tableau correspond to different subcases in a proof. Obviously, Def. 3.2(iii) is a means to produce a contradiction in a subcase/branch. A tableau proof is finished when this has been achieved for all branches. The following terminology is standard:

Definition 3.3. *In a tableau T for a set Φ of sentences a branch B is* closed *iff $B \cup \Phi$ contains a pair $\phi, \neg\phi \in \mathcal{L}_{\Sigma^*}$ of complementary formulas, or false; otherwise, it is* open*. A tableau is closed if* all *its branches are closed.*

[2] When no confusion can arise, branches are frequently identified with the set of their nodes (formulas).

A tableau proof *for (the unsatisfiability of) a set* $\Phi \subset \mathcal{L}_\Sigma$ *of sentences is a closed tableau* T *for* Φ*.*

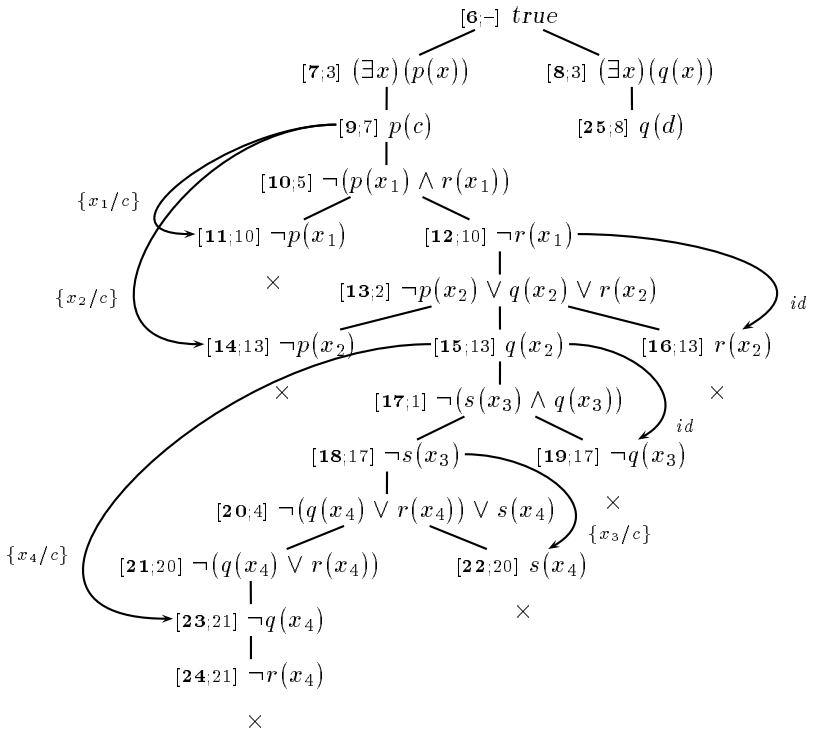


Figure 2. Partial tableau proof for Φ from Example 3.1.

Example 3.1. We are now in a position to formalize the introductory example from Section 3.1. Sets P, Q, R, S are represented by unary predicates p, q, r, s: their characteristic functions. Then, over the signature $\Sigma = \langle \{p, q, r, s\}, \{\} \rangle$, the claim holds if and only if the set Φ consisting of $\mathcal{L}_\Sigma$-sentences (1) $\neg(\exists x)(s(x) \wedge q(x))$, (2) $(\forall x)(\neg p(x) \vee q(x) \vee r(x))$, (3) $(\exists x)(p(x)) \vee (\exists x)(q(x))$, (4) $(\forall x)(\neg(q(x) \vee r(x)) \vee s(x))$, (5) $\neg(\exists x)(p(x) \wedge r(x))$ is unsatisfiable.

Figure 2 shows a tableau T for Φ. The nodes of the tableau are numbered starting from 6 (the numbers 1–5 refer to the formulas in Φ); an expression $[\mathbf{i}; j]$ is in front of the i-th node N_i, where j signifies that N_i stems from an expansion rule applied to N_j (respectively, to formula (j) in Φ).

All branches of T except the rightmost one can be closed by virtue of the closure rule; a closure is indicated by an arc between its complementary literals, labelled with the required MGU. T is not yet a tableau proof for Φ, but it can be extended

to one by adding a copy of the subtableau with root node 17 below node 25 and instantiating the free variables in that copy with d (instead of c).

3.2.4. Tableau Semantics and Soundness
Since our goal is to use tableaux as a framework for formal proofs, we require to extend semantics from formulas to tableaux. Our guideline here is to ensure that a tableau for Φ is closed iff Φ is unsatisfiable. We fixed already that a tableau is thought of as the disjunction of its branches which in turn are considered as conjunctions of their labels. By a standard argument then, the equivalence above is reduced to the question whether the tableau construction rules leave tableau satisfiability unaltered. This is a routine matter for all but the δ-rule which requires some care. Recall that two optimizations were incorporated into this rule (Section 3.2.2): (i) the variables of the Skolem term are restricted to the free variables of δ, (ii) the leading function symbol sko_δ of the Skolem term is not unique in a tableau proof. Tableau semantics must be carefully chosen to reflect these restrictions. To meet (i) it suffices to treat free variables in a tableau essentially as if they were universally quantified.

Definition 3.4. *A tableau T for $\Phi \subset \mathcal{L}_\Sigma$ is* satisfiable *if there is a structure $\mathbf{M}$ of Φ such that for every variable assignment μ there is a branch B of T with $(\mathbf{M}, \mu) \models B$. In that case we say that $\mathbf{M}$ is a model of T, denoted by $\mathbf{M} \models T$.*

For (ii) it is important that Skolem function symbols are interpreted in the "right" way. The most elegant way to achieve this, is to define formula semantics with respect to only such interpretations—let us call them *canonical interpretations*. Of course, one needs to show then that each satisfiable formula can be satisfied by a canonical interpretation.

Definition 3.5. *A term structure $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ is* canonical *iff for all variable assignments μ and $\delta(x) \in \mathcal{L}_{\Sigma^*}$: if $(\mathbf{M}, \mu) \models \delta(x)$ then $(\mathbf{M}, \mu) \models \delta_1(sko_\delta(x_1, \ldots, x_n))$, where $x_1, \ldots, x_n$ are the free variables in δ.*

Lemma 3.1 ([7])**.** *Given a signature Σ, if the set $\Phi \subset \mathcal{L}_\Sigma$ of sentences is satisfiable, then there is a canonical structure $\mathbf{M}^*$ over Σ^* such that $\mathbf{M}^* \models \Phi$.*

Corollary 3.1. *Let $\mathbf{M}^*$ be a canonical structure over Σ^*, μ a variable assignment, and $\phi \in \mathcal{L}_{\Sigma^*}$; and let ϕ' be constructed from ϕ by (a) replacing a positive occurrence of some $\delta(x)$ in ϕ by $\delta_1(sko_{\delta(x)}(x_1, \ldots, x_n))$, or by (b) replacing a negative occurrence of $\overline{\delta(x)}$ in ϕ by $\overline{\delta_1(sko_{\delta(x)}(x_1, \ldots, x_n)}$, where $x_1, \ldots, x_n$ are the free variables in $\delta(x)$. Then $(\mathbf{M}^*, \mu) \models \phi$ implies $(\mathbf{M}^*, \mu) \models \phi'$.*

Lemma 3.2. *Any tableau for a satisfiable set of $\mathcal{L}_\Sigma$-sentences is satisfiable.*

Proof. By definition of tableaux there is a sequence $T_1, \ldots, T_m$ $(m > 0)$, where $T = T_m$ and T_1 is the initial tableau whose single node is *true*, and where T_{i+1}

is constructed from T_i by applying a single tableau expansion or closure rule. By Lemma 3.1, the input set is satisfied by a canonical structure $\mathbf{M}^*$ over Σ^*. By induction on m one proves that $\mathbf{M}^*$ satisfies all of $T_1, \ldots, T_m$ and hence T. The induction step is trivial and all cases, but the δ-rule case are straightforward (see [18] for a detailed proof). The latter, however, follows from the corollary. □

Now assume we have a closed tableau T for a set of $\mathcal{L}_\Sigma$-sentences Φ. Obviously, no structure and variable assignment can satisfy a closed branch, so T is unsatisfiable. By the preceding lemma, Φ is unsatisfiable as well. This proves:

Theorem 3.1 (Soundness)**.** *If there is a tableau proof for a set $\Phi \subset \mathcal{L}_\Sigma$ of sentences, then Φ is unsatisfiable.*

3.3. From Calculus to Proof Procedure

Tableau soundness gives the desirable property of tableaux with unification that a closed tableau for Φ signifies validity of $\overline{\Phi}$. Remains the question whether for *all* valid sentences a tableau proof exists and, if this is the case, how it can be found. While the first question is answered affirmatively in Section 3.4, for the second one a fully satisfactory answer is not yet available. Let me point out why it is a difficult problem.

It is straightforward to derive a tableau proof procedure from Definition 3.2, but note that such a procedure is indeterministic: usually, a great number of rules is applicable to any given tableau. More precisely, one first must select a branch B, where a rule is applied, then decide whether an expansion rule or a closure rule is used; in the first case one must choose a formula $\psi \in B \cup \Phi$, in the second case a pair of literals on B. Let us refer to these kinds of indeterminism with the phrases *select branch*, *select formula*, *select pair*, and *select mode* in the following. Making these choices deterministic in an arbitrary way almost certainly results in an incomplete proof procedure.

In Figure 3, for example, the γ-formula is always preferred for expansion rule application, delaying expansion of the inconsistent second formula indefinitely. In an obvious way, the formula $Q \wedge \neg Q$ is treated *unfair*. This motivates the following definition.

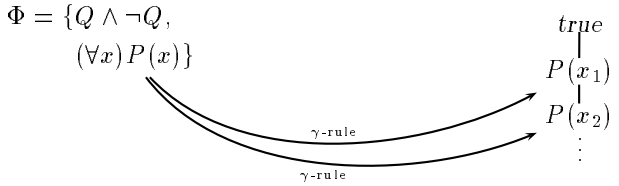


Figure 3. Incompleteness caused by unfair *select formula*.

Definition 3.6. *The construction of a sequence $(T_n)_{n>0}$ of tableaux for $\Phi \subset \mathcal{L}_\Sigma$, where T_n is obtained from T_{n-1} by applying a tableau expansion rule, is* fair *if the following holds for all branches B in the infinite tableau that is approximated by that sequence: (1) all α, β, and δ occurring on B or in Φ were used to expand B (by applying the appropriate expansion rule); (2) all γ occurring on B or in Φ were used infinitely often to expand B (by applying the γ-rule).*

One may construct a fair sequence of tableaux for any set of sentences. Combining fair application of the expansion rule with fair application of the closure rule, however, is a difficult problem, because tableaux with unification are *destructive*:

Definition 3.7. *A tableau calculus is* non-destructive *if all tableaux that can be constructed from a given tableau T contain T as an initial subtree; otherwise the calculus is* destructive.

For example, at first sight it might seem to be a good idea to apply the closure rule in a "greedy" manner that is as early as possible. Not so. One problem is that several pairs of closure literals (with incompatible MGUs) may compete, but this is not all. In Figure 4, independently from which branch is closed first, the variable x_1 gets "used up" by a substitution that blocks closure of the other branch. Of course, a second free variable instance of the γ-formula may be created, but then the same happens one level below etc.

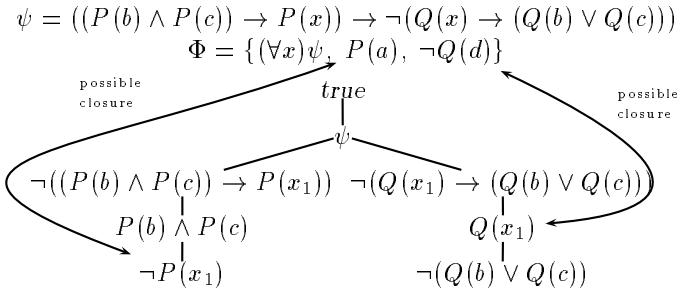


Figure 4. Incompleteness caused by unfair *select mode*.

A non-destructive tableau procedure does not suffer from this problem, because it is (trivially) *proof confluent*:

Definition 3.8. *A tableau proof procedure is* proof confluent *if from every tableau for an unsatisfiable set of sentences a closed tableau can be constructed.*

A destructive tableau proof procedure still might be proof confluent, but as witnessed by the previous example, it might as well be not. A deterministic tableau proof procedure which is not proof confluent is incomplete. At the present time, no practical, proof confluent destructive tableau proof procedure is known (there is

hope, however, see the discussion at the end of Section 3.3). Therefore, it is worth discussing possible ways around the problem.[3]

An obvious way to enforce proof confluency of tableaux with unification consists in delaying any application of the closure until all tableau branches can be closed simultaneously by a suitable substitution. This is the path chosen in Fitting's text book [18]—it has its inefficencies: first, one cannot discard closed branches until the proof is essentially finished which might lead to storage problems, and second, after each expansion rule the whole tableau must be tested for closure which is very redundant.

Another option for obtaining a complete tableau proof procedure comes from the observation that its indeterminism is locally finite—from each tableau only a finite number of successors can be constructed. Thus one can turn each tableau into a finitely branching OR node of an AND-OR search tree which may then be searched in a breadth first manner. This approach is impractical because of space requirements.

Stickel [59] suggested to replace breadth first search by *d*epth *f*irst search with backtracking and *i*terative *d*eepening of the search depth (DFID search) which has only a small overhead in run time as compared to breadth first search, but is a lot more space efficient [33]. Tableau proof search procedures based on DFID turn out to be elegantly and efficiently implementable in logic programming languages [59, 3, 8].

To increase efficiency of search it is important to get rid of as many OR nodes as possible. In the DFID setup this means to minimize the amount of backtracking. It is obvious that one may choose any deterministic strategy for *select branch* as all branches need to be closed eventually. In addition, it is not difficult to implement a fair strategy for *select formula* [18]. This leaves the—destructive—branch closure. Ideally, one would like to have a fair selection rule not only for formula selection, but for closure selection as well, thus yielding a proof confluent proof search procedure for tableaux with unification.

3.4. Tableau Completeness

The preceding discussion shows that it is difficult to ensure completeness of tableaux with unification for a deterministic proof search procedure. On the other hand, it is not too difficult to show mere existence of a closed tableau for each unsatisfiable set of sentences. The presentation of the latter result closely follows [7].

It is convenient to work with a data structure that slightly abstracts from tableau branches: so-called Hintikka sets (named after their inventor [30]) may contain an infinite number of formulas whose order is irrelevant. A model can be immediately constructed for any Hintikka set.

[3] Another reason is that some of the more important refinements of the tableau calculus are not proof confluent already on the propositional level. This discussed in Section 4.3 below.

Definition 3.9. *A set $H \subset \mathcal{L}_{\Sigma^*}$ of sentences is a* Hintikka set *if it satisfies the following conditions: (1) false $\notin H$ and there are no complementary literals in H. (2) If $\alpha \in H$, then all α_i are in H. (3) If $\beta \in H$, then some β_i is in H. (4) If $\gamma(x) \in H$, then $\gamma_1(t) \in H$ for all ground Σ^*-terms t. (5) If $\delta(x) \in H$, then $\delta_1(t) \in H$ for some ground Σ^*-term t.*

Lemma 3.3 (Hintikka)**.** *Every Hintikka set is satisfiable.*

Proof. A term model $\mathbf{M} = \langle \mathbf{D}, \mathbf{I} \rangle$ of H is defined by: $t^{\mathrm{I}} = t$ for all $t \in \mathbf{D}$, and $p^{\mathrm{I}}(t_1, \ldots, t_k) = \mathit{true}$ iff $p(t_1, \ldots, t_k) \in H$ for ground atoms $p(t_1, \ldots, t_k)$ over Σ^*. By induction on the structure of formulas in H it is easy to prove that $\mathbf{M} \models H$. □

Theorem 3.2 (Completeness)**.** *If the set $\Phi \subset \mathcal{L}_\Sigma$ of sentences is unsatisfiable, then there is a tableau proof for Φ.*

Proof. Let $(T_n)_{n>0}$ be a fair sequence of tableaux for Φ starting with the tableau consisting of the single node *true*; this sequence approximates the infinite tree T_∞. We define a particular ground substitution σ_∞ as follows: let $(B_k)_{k>0}$ be an enumeration of the branches of T_∞ and $(\phi_i)_{i>0}$ an enumeration of the γ-formulas in T_∞. For every γ-formula ϕ_i, if ϕ_i occurs on B_k let x_{ijk} be the new variable introduced by the j-th application of the γ-rule to ϕ_i on B_k. Finally, let $(t_j)_{j>0}$ be an enumeration of all Σ^*-ground terms.

To ensure that B_k defines a model, the instances of ϕ_i on $B_k\sigma_\infty$ must "cover" all ground terms t_j. It suffices to choose $\sigma_\infty(x_{ijk}) = t_j$ for all $i, j, k > 0$.

By construction of σ_∞ and fairness of T_∞, if B is a branch in T_∞ and $B\sigma_\infty$ is open, then $B\sigma_\infty \cup \Phi$ is a Hintikka set and so Φ is satisfiable. This contradicts the assumption, hence $T_\infty\sigma_\infty$ is closed. $T_\infty\sigma$ is a finitely branching tree and the distance of all formulas involved in closures to the root node is finite. Then, by König's Lemma[4], there is an $n > 0$ such that the finite tableau $T_n\sigma_\infty$ is closed.

In general, σ_∞ is not a *most general* unifier of complementary literals used in closures and cannot be used in an MGU closure rule application to T_n. Therefore, it remains to show that σ_∞ can be suitably decomposed: $\sigma_\infty = \sigma \circ \sigma_r \circ \sigma_{r-1} \circ \cdots \circ \sigma_1$, where σ_i is a most general closing substitution for the instance $B_i\sigma_1\sigma_2 \ldots \sigma_{i-1}$ of the i-th branch in T_n $(0 < i < r + 1)$; σ is the part of σ_∞ not actually needed to close T_n. The σ_i are constructed inductively:

Let $\sigma'_1 = \sigma_\infty$. For $1 < i < r + 1$, let σ_i be a most general substitution such that (1) σ'_{i-1} is a specialization of σ_i (there is a substitution σ'_i such that $\sigma'_{i-1} = \sigma'_i \circ \sigma_i$) and (2) σ_i is a closing substitution for $B_i\sigma_1\sigma_2 \ldots \sigma_{i-1}$. Now σ_i is a most general *closing* substitution of $B_i\sigma_1\sigma_2 \ldots \sigma_{i-1}$. Otherwise, there is a closing substitution σ''_i being more general than σ_i. The is-more-general relation is transitive, hence σ''_i is more general than σ'_{i-1} in contradiction to σ_i being already a suitable most general substitution. Finally, let $\sigma = \sigma'_r$. □

[4] "A tree that is finitely branching but infinite must have an infinite branch." A proof is, for example, in [18].

It suffices to apply the appropriate expansion rule exactly once to each α, β, or δ-formula on each branch to obtain a Hintikka set from a fairly constructed sequence of tableaux. This has the practically relevant consequence that only to γ-formulas must a rule be applied more than once per branch.

4. Clause Tableaux

In the present section a number of refinements of the tableau procedure are introduced. For a number of reasons, these refinements are merely discussed on the clause level:

- Simplified notation leads to easier detection of new refinements
- Efficient implementability
- Completeness proofs stay managable
- Comparability (most deduction procedures implemented on clause level)

Restricting attention to the clause level implies some limitations as well:

- Some applications (such as software verification) expect proofs on non-clausal level: backtranslation from clauses can be tricky
- For some non-classical logics a clause normal form is unknown

So there is considerable incentive to generalize the following results (partially this has been done, for example, in [25]), but I think the material is more accessible in the present, syntactically limited form.

4.1. Normal Form Computation

How first-order sentences are efficiently transformed into clause sets is shown, for example, in [49, 43].

4.2. Clause Tableau Proofs, Soundness, Completeness

4.2.1. Clause Tableaux

Let us start by stating suitably simplified versions of Definition 3.2 and 3.3.

Definition 4.1. *Let Σ be a first-order signature. A* clause tableau *(over Σ) is a finitely branching tree whose nodes are literals from $\mathcal{L}_{\Sigma^*}$. Given a clause set S from $\mathcal{L}_\Sigma$, a clause tableau for S is constructed by a (possibly infinite) sequence of applications of the following rules:*

(i) *The tree consisting of a single node labeled with* true *is a tableau for S (initialization rule).*

(ii) *Let T be a tableau for S, B a branch of T, and $L_1 \vee \cdots \vee L_r$ an instance of $C \in S$. If the tree T' is constructed by extending B with r new subtrees and the nodes of the new subtrees are labeled with L_i, then T' is a tableau for S (extension rule).*

(iii) *Let T be a tableau for S, B a branch of T, and L and L' literals in S. If L and $\overline{L'}$ are unifiable with MGU σ, and T' is constructed by applying σ to all literals in T (i.e., $T' = T\sigma$), then T' is a tableau for S (closure rule).*

Definition 4.2. *In a clause tableau T for a set S a branch B is* closed *iff B contains a pair of complementary literals; otherwise, it is* open*. A tableau is closed if* all *its branches are closed.*

A clause tableau proof *for (the unsatisfiability of) a clause set $S \subset \mathcal{L}_\Sigma$ is a closed clause tableau T for S.*

Example 4.1. A clause form of the formula set in Example 3.1 consists of clauses (1) $\neg s(x) \vee \neg q(x)$, (2) $\neg p(x) \vee q(x) \vee r(x)$, (3) $p(c) \vee q(d)$, (4) $\neg q(x) \vee s(x)$, (5) $\neg r(x) \vee s(x)$, (6) $\neg p(x) \vee \neg r(x)$. Observe that the nodes are a subset of the nodes in Figure 2. Clause (5) is not used.

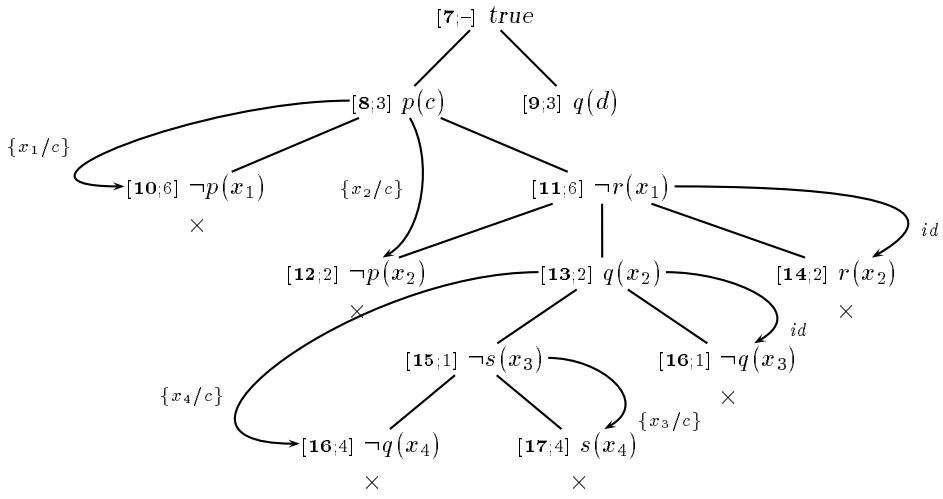


Figure 5. Partial clause tableau proof for S from Example 4.1.

Clause tableaux mainly constitute a syntactic simplification of full first-order tableaux. The main properties of the calculus are the same, in particular the discussion in Section 3.3 applies to them as well.

In contrast to the full first-order case the extension and closure rule only use literals and clauses from the input set. This simplifies some definitions.

4.2.2. Soundness and Completeness

Soundness of clause tableaux follows immediately from Theorem 3.1 by observing that clauses are particular first-order formulas and extension rule 4.1.(ii) can be composed of several applications of rule 3.2.(ii).

Completeness could be obtained easily by suitable simplification of the proof of Theorem 3.2, but in the clausal case a more modular approach is useful. Following Robinson [52], lifting a ground proof to a first-order proof is separated from proving ground completeness of a calculus. The advantage is that the lifting part is similar for all following tableau refinements and must at most be sketched each time. So it is sufficient to concentrate on ground completeness. Abstraction from first-order issues greatly simplifies completeness proofs of the more complicated calculi that follow.

Theorem 4.1 (Lifting)**.** *Let S be an unsatisfiable clause set, $\overline{S}$ an unsatisfiable set of ground instances of S and $\overline{T}$ a closed clause tableau for $\overline{S}$. Then there is a closed clause tableau T for S and a substitution τ such that $\overline{T} = T\tau$.*

Proof. The main technical difficulty of this proof is that in Definition 4.1.(iii) only MGUs are to be used whereas $\overline{S}$ may contain arbitrary ground instances of clauses. The following property of MGUs is needed:

$$\text{If } T' \text{ is a clause tableau, } \tau \text{ a substitution such that } T'\tau \text{ is closed, and } \rho \text{ an MGU that closes any branch of } T', \text{ then } T'\rho\tau = T'\tau. \tag{4.1}$$

Proof of (4.1): by definition of an MGU and as τ closes T', there is τ' with $\rho\tau' = \tau$. MGUs can be assumed to be idempotent, so $T'\rho\tau = T'\rho\rho\tau' = T'\rho\tau' = T'\tau$.

Back to the main proof, let T^0 be constructed exactly as $\overline{T}$ but for each extension step with $\overline{C} \in \overline{S}$ used in $\overline{T}$, take instead an instance of the clause $C \in S$ of which $\overline{C}$ is a ground instance. Obviously, $T^0\tau = \overline{T}$ for a suitable ground substitution τ.

If B is an arbitrary open branch of T^0, then it is closed by τ, so there is an MGU ρ that closes B and rule 4.1.(iii) is applicable to obtain a clause tableau $T^1 = T^0\rho$. By (4.1), $T^1\tau = T^0\tau = \overline{T}$. Repeating this argument in a straightforward induction over the number n of open branches in T^0 yields a closed clause tableau $T = T^n$ such that $T\tau = \overline{T}$. □

In the proof the sequence of branch closures was arbitrary which shows independence of the *select branch* strategy.[5]

In the following it is sufficient to prove ground completeness for various instances X of clause tableau restrictions:

Theorem 4.2 (Ground Completeness Schema)**.** *If the finite ground clause set S is unsatisfiable, then there is a X-tableau proof for S.*

We could proceed to prove ground completeness of unrestricted clause tableaux right now, but in following sections ground completeness of various restrictions of clause tableaux is proven of which completeness of the unrestricted calculus is an immediate consequence.

[5] When *select clause* is arbitrary, but fair, the theorem still holds in the weakened form that there is a clause tableau T for S such that $T\tau$ can be homeomorphically embedded into $\overline{T}$.

Theorem 4.3 (Completeness)**.** *If the clause set S is unsatisfiable, then there is a clause tableau proof for S.*

Proof. Herbrand's Theorem 2.2 provides a finite, unsatisfiable set $\overline{S}$ of ground instances of S. By Theorem 4.2 there is a closed ground clause tableau $\overline{T}$ for $\overline{S}$ and, by Theorem 4.1, there is a closed clause tableau for S. □

As announced already, the next goal is to find complete restrictions of clause tableaux. It is sufficient to provide a suitable instance of Theorem 4.2, whenever a ground X-tableau proof $\overline{T}$ lifts to a first-order X-tableau proof T. It is usually sufficient to check that the proof of Theorem 4.1 can be used unaltered.

From the point of proof search, restricting the tableau calculus means to exclude certain choices in *select clause* and *select pair* and to fix *select branch* in some way.

4.3. Connections

Connection conditions have been pioneered by Davydov [17], Bibel [11, 12], and Andrews [1].

4.3.1. Strong Connections

A major drawback of the tableau calculus is that the extension rule 4.1.(ii) is applied completely unguided which can clutter up tableaux with many nodes that do not contribute to a proof.

Example 4.2. Consider the two clause tableaux for $S = \{p(x) \vee q(x), r(x) \vee s(x), \neg p(a), \neg q(a), \neg r(b), \neg s(b)\}$ displayed in Figure 6. The tableau on the right constitutes a minimal proof, while the second extension step in the tableau on the left is completely unrelated to the initial step.

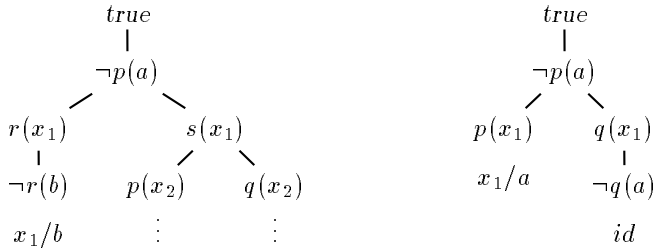


Figure 6. Redundant nodes in a tableau.

Definition 4.3 ([36])**.** *A* connection tableau *is a clause tableau in which every non-leaf node L (except* true*) has $\overline{L}$ as one of its immediate successors.*

The tableau on right in Figure 6 is a connection tableau, the tableau on the left is not. It is excluded by the connection restriction.

Connection tableaux are complete, but the proof is deferred until the next section. The definition of connection tableaux implies that when $T \neq \mathit{true}$ at least one of the new branches generated by the tableau extension rule can be closed. This suggests an *implementation-oriented* definition of connection tableaux obtained from Definition 4.1 by changing the first two rules:

(i′) For any instance $L_1 \vee \cdots \vee L_r$ of $C \in S$ the tree constructed by extending *true* with r new subtrees with nodes L_i is a connection tableau for S.

(ii′) Let T be a tableau for S, B a branch of T ending with L, $L_1 \vee \cdots \vee L_r$ an instance of $C \in S$. If $\overline{L}$, L_i (where $i \in \{1, \ldots, r\}$) are unifiable with MGU σ and the tree T' is constructed by extending B with r new subtrees, where the nodes of the new subtrees are L_i, then $T'\sigma$ is a connection tableau for S. This is called an *extension step*.

Closures of open branches are unchanged and called *reduction step*. It is important to note that while clause tableaux are independent of *select clause*, connection tableaux are not:

Proposition 4.1. *Ground connection tableaux are not proof confluent.*

Proof. Consider $S = \{p, \neg p, q\}$ and let q be the initial clause. It impossible to make any further extension step, although S is clearly unsatisfiable. (Examples independent of the initial clause can be found in [35].) □

4.3.2. Weak Connections

The extension rule (ii′) of connection tableaux has a natural weakening:

(ii″) Let T be a tableau for S, B a branch of T *containing* L, $L_1 \vee \cdots \vee L_r$ an instance of $C \in S$. If $\overline{L}$, L_i (where $i \in \{1, \ldots, r\}$) are unifiable with MGU σ and the tree T' is constructed by extending B with r new subtrees, where the nodes of the new subtrees are L_i, then $T'\sigma$ is a clause tableau for S. (This is called a *weakly connected extension step*.)

Let us call the resulting calculus *weak connection tableaux*.

Definition 4.4. *A clause set is* minimally unsatisfiable (mu) *when it is unsatisfiable and each of its proper subsets is satisfiable. A clause is* relevant *in S when it is contained in a mu subset of S.*

It can be shown that weak connection tableaux are proof confluent provided that *select clause* is implemented in a fair manner and the initial clause is relevant. Unfortunately, testing for membership in a mu set is as expensive as testing unsatisfiability itself. This limits the usefulness of weak connection tableaux in practice, but in Section 4.5 a slight relaxation will build the basis of a whole class of interesting calculi which are proof confluent regardless of the initial clause.

4.4. Regularity

One of the most important strategies in resolution theorem proving is *subsumption*. In the tableau-based theorem proving it can be approximated by *regularity*:

Definition 4.5. *A clause tableau is* regular *when all of its branches contain at most one occurrence of the same literal.*

Example 4.3. Regularity can help to avoid applying substitutions that lead to redundant proofs. Consider the tableau for $S = \{p(0), \neg p(x) \vee p(s(x)), \neg p(s(s(0)))\}$ in Figure 7. The first possible substitution for the middle branch renders the right branch irregular and is thus avoided.

Implementing regular tableaux is not straightforward, because an admissible closure substitution can potentially unify as well formerly different literals on branches closed already. For efficiency reasons one discards closed branches immediately, so there must be a mechanism to exclude such critical substitutions. It was suggested in [36] to create an inequality constraint of the form $t_1 \neq t'_1 \vee \cdots \vee t_m \neq t'_m$, whenever two unifiable literals $(\neg)p(t_1, \ldots, t_m)$ and $(\neg)p(t'_1, \ldots, t'_m)$ are encountered on one branch. Then substitutions are applied to constraints as well and must ensure their satisfiability. In the example above, the second extension step generates the constraint $s(0) \neq s(x_2)$ which is not satisfied by $x_2/0$.

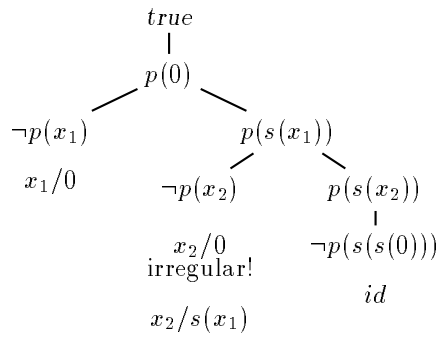


Figure 7. Advantage from regularity.

The following standard lemma is needed in the proof of of ground completeness of regular connection tableaux. Its easy proof is given, for instance, in [39, Lemma 2.3.2, p. 63]. The completeness theorem was first proven (differently) in [35]. The present proof is due to [26].

Lemma 4.1. *Let S be a mu ground clause set with $C \in S$, $D \subseteq C$, and $S_D = (S - \{C\}) \cup \{D\}$. Then for any mu subset S'' of S_D: (i) $D \in S''$; (ii) $D \not\subseteq D''$ for all $D \neq D'' \in S''$.*

Theorem 4.4. *If the finite ground clause set S is unsatisfiable, then there is a regular connection tableau proof for S.*

Proof. We show by induction on the number k of literal occurrences in S: for any relevant non-unit clause $C_j \in S$, there is a closed regular connection tableau for S whose initial step uses C_j. When there is no such clause there must be a mu subset of unit clauses in S; it is trivial to find a regular connection tableau proof for such a set.
$k \in \{0, 1, 2\}$: either S is satisfiable or it contains only unit clauses or the empty clause and the claim is trivially satisfied.
$k > 2$: (see Figure 8) let $C_j = L_1 \vee \cdots \vee L_i \vee \cdots \vee L_n$ be a relevant non-unit clause in S and let T be the regular connection tableau resulting from an initial step with C_j.

For all $i \in \{1, \ldots, n\}$ let $S_{L_i} = (S - \{C_j\}) \cup \{L_i\}$. By Lemma 4.1.(i), $C'_j = L_i$ is contained in an mu subset S'_i of S_i. Hence, $\overline{L_i}$ occurs in a clause C^i of S'_i. Moreover, S'_i contains less literals than S.

If C^i is a unit clause, then the i-th branch of T can be closed immediately, resulting in a regular connection tableau. Otherwise, applying the induction hypothesis on C^i and S'_i yields a closed regular connection tableau T_i for $S_i \subseteq S'_i$, where the first extension step uses C^i.

By Lemma 4.1.(ii), L_i occurs at most in C'_j and is therefore only used in unit extension steps in T_i (highlighted by boldface type in the figure). As shown in the figure, each T_i is glued together with T at L_i maintaining connectedness. In the resulting tableau irregularity can at most occur with the L_i. But as L_i occurs on top of each T_i (circled occurrence) the unit extension steps with L_i simply can be replaced by reduction steps with the circled occurrence of L_i. As $S_i - \{C'_j\} \subseteq S$, the result is a regular connection tableau for S. □

4.5. Orderings and Selection Functions

4.5.1. Redundancy and Saturation in Tableaux

Let us take up the theme expressed in the regularity restriction, namely to avoid redundancy in tableau proofs.

Any open branch B in a ground clause tableau T for S, or equivalently, any consistent set of ground literals B defines an *interpretation* $\mathbf{I}_B$ on S via $\mathbf{I}_B \models L$ iff $L \in B$. A natural notion of redundancy then would be the following: Clause $C \in S$ is redundant on B when $\mathbf{I}_B \models C$. The idea is that tableaux being a refutation calculus it is not interesting to use clauses that are already modelled by branch B. Such clauses are not used in extension steps. Irregular extension steps and tautologous clauses are redundant in this sense, but a stronger notion of redundancy is desirable.

Definition 4.6. *An open clause tableau branch B has a* saturation *wrt S iff it has an extension $\overline{B} \supseteq B$ such that $\boldsymbol{I}_{\overline{B}} \models S$.*

It is, of course, not realistic to consider *all* possible extensions of a branch (the empty branch, for example, always has a saturation when S is satisfiable), so we

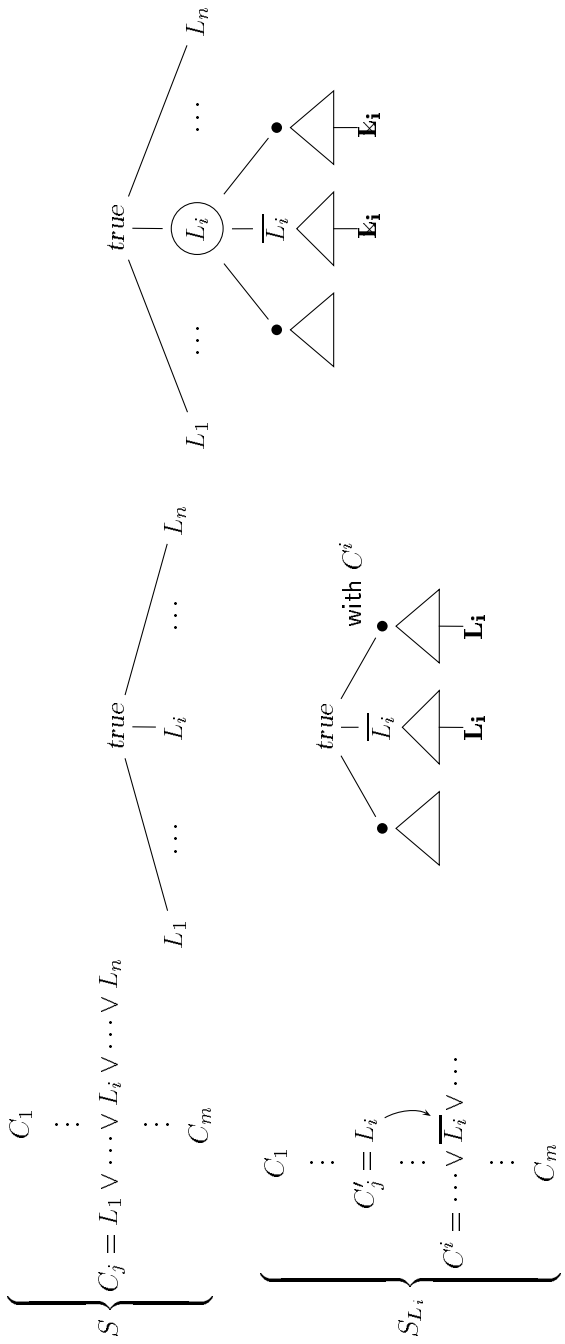


Figure 8. Illustration of the proof of Theorem 4.4

check only one of them to guide tableau extension. Informally, we will consider extensions of B that contain literals from all clauses of S not yet satisfied by $\mathbf{I}_B$.

4.5.2. Tableaux with Selection Function
Let f be a *selection function* on clauses: a function mapping each clause into a (possibly empty) subset of its literals.

$$\overline{B}_f = B \cup \{f(C) \mid C \in S, \mathbf{I}_B \not\models C\} \tag{4.2}$$

If $\overline{B}_f$ is consistent and all clauses with no selected literals were used on B, then $\mathbf{I}_{\overline{B}_f} \models S$ by construction and proof search can be stopped here. In general, however, $\overline{B}_f$ does not induce an interpretation, because it may contain complementary literals. In this case, one of the clauses not yet satisfied by $\mathbf{I}_B$ is selected for extension whose selected literal(s) contribute to a contradiction in $\overline{B}_f$. Formally, let $\overline{f(C)}$ denote the set of complements of literals selected by f in C. Then extension steps (Definition 4.1.(ii)) are restricted to clauses in

$$\{C \mid C \in S, \mathbf{I}_B \not\models C \text{ and } (\overline{f(C)} \cap \overline{B}_f \neq \emptyset \text{ or } f(C) = \emptyset)\} \tag{4.3}$$

By definition, if $L \in \overline{f(C)} \cap \overline{B}_f$, either $L \in B$ or $L \in f(D)$ for some clause $D \in S$. In the first case the extension is a *weak connection step* in the sense of (ii″) on page 21. The second case and when no literal is selected is called a *restart step* (and C a *restart clause*) to emphasize that the tableau proof bears no direct connection with the current branch. The first clause used in a tabelau is always a restart step. No new restart clauses are added once f is fixed, they can be computed in advance.

Example 4.4. Consider the clause set $S = \{\neg q \vee \underline{\neg s}, \neg r \vee \underline{s}, p \vee q \vee \underline{r}, \underline{\neg p}\}$ in which the lexicographically largest literal is selected (these are underlined). Initially, $B = \{true\}$ and $\overline{B}_f = \{\neg s, s, r, \neg p\}$. The first two clauses are the only restart clauses of which the first is selected (see Figure 9). Branch $B = \{\neg q\}$ implies $\overline{B}_f = B \cup \{s, r, \neg p\}$ which models S. On the other branch $B = \{\neg s\}$ one has $\overline{B}_f = B \cup \{s, r, \neg p\}$, so an extension step (the only one) with the second clause is possible. The only open branch is now $B = \{\neg s, \neg r\}$ with $\overline{B}_f = B \cup \{r, \neg p\}$ and only extension with the third clause is allowed. The first open branch, $\{\neg s, \neg r, p\}$ is closed by a further extension while the last open branch $B = \{\neg s, \neg r, q\}$ yields $\overline{B}_f = B \cup \{\neg p\}$ and thus the second model of S.

Theorem 4.5. *If the finite ground clause set S is unsatisfiable, then for any selection function f there is a tableau proof with selection function f for S.*

Proof. Assume there is a tableau with selection function f and an open branch B in which all possible extension steps were made. As S is finite, B is finite as

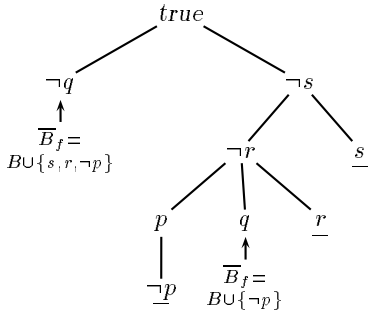


Figure 9. Tableau with selection function.

well. We claim that $\overline{B}_f$ is consistent from which $\overline{B}_f \models S$ follows by construction (in particular, all clauses with no selected literal are satisfied). If $\overline{B}_f$ is inconsistent there is a literal $L \in \{f(C) \mid C \in S, \mathbf{I}_B \not\models C\}$ such that $\overline{L}$ occurs (I) in B or (II) in $f(D)$ for some clause D not yet satisfied by B. In case (I) a weakly connected extension step is possible on B in case (II) a restart step is admissible. Either way, the assumption that all possible extension steps were made on B is contradicted. □

The proof is independent of the sequence of extension steps chosen, so tableaux with selection function are *proof confluent*. Moreover, a slight generalization of the proof shows that completeness is retained even when f is changed during tableau construction.

4.5.3. Related Calculi

A number of recently suggested restrictions of clause tableaux can be considered as special cases of tableaux with selection function, for example, ordered tableaux [32, 25].

Definition 4.7. *A ground* atom (A-)ordering *is a binary relation* $<$ *on atoms which is irreflexive and transitive.*

A-orderings give a complete tableau restriction which can be expressed via selection functions as follows: simply use $f_<(C) = \{L \mid L \text{ maximal in } C \text{ wrt } <\}$ and delete the requirement $\mathbf{I}_B \not\models C$ in (4.2,4.3). The latter relaxation plus the requirement $|f(C)| > 0$ gives the version of tableaux with selection function discussed in [27]. On the other hand, [46] show that tableaux with selection functions can be modified to accomodate strongly connected extension steps.

4.5.4. First-Order Issues

Lifting is straightforward for tableaux with selection function f provided that f lifts. Call f *stable wrt substitutions* if $f(C\sigma) \subseteq f(C)\sigma$ for all clauses C and substitutions σ. For A-orderings this translates into the requirement $p < q$ implies $p\sigma < q\sigma$ for all substitutions σ and atoms p, q.

Implementation of first-order tableaux with selection function generates similar problems as regularity. The first point concerns the condition $\mathbf{I}_B \not\models C$ in (4.2,4.3). Unless B contains ground literals this imposes hardly any restriction in the first-order case, but one may generate inequality constraints: let $\sigma = \{x_1/t_1, \ldots, x_r/t_r\}$ be an MGU such that $\mathbf{I}_{B\sigma} \models C\sigma$; for each such σ a constraint of the form $x_1 \neq t_1 \vee \cdots \vee x_r \neq t_r$ is added. Further constraints to be added are derived from (4.3) and take on the form $L \in f(C)$ [45, 27].

Checking these latter constraints for satisfaction can be expensive (NP-complete), so one may decide to suppress their generation. The resulting calculus might be called *tableaux with input selection function* [27], because the selection restriction is only enforced on clauses that serve as input for extension steps, but not on instances of clauses appearing in a tableau. This has another advantage: it was noted after the proof of Theorem 4.5 that the selection function may be arbitrarily changed during tableau construction. This implies immediately that selection functions need not be stable wrt substitutions in input selection tableaux, rather, stability wrt variable renamings is sufficient [27].

4.6. Hyper Tableaux

Hyper calculi share the feature that several deduction steps are combined into one. This yields a speed-up in proof search, but the main advantage is that some intermediate results are not computed in the first place and this can limit the search space considerably. In the ground case one stipulates a similar condition as (4.3) on extending clauses saying that *all* selected literals of an extending clause must be weakly connected to the current branch, formally:

$$\{C \mid C \in S, \mathbf{I}_B \not\models C \text{ and } \overline{f(C)} \subseteq B\} \tag{4.4}$$

A ground clause tableau constructed with this restriction on extension steps (Definition 4.1.(ii)) is called a *hyper tableau*. An extension step is combined with as many closure rule applications as there are selected literals in the extending clause.

In general, hyper tableaux are not complete (for example, when *all* literals are selected), but completeness is regained for *consistent selection functions* that do not both select a literal and its complement in any input clause.[6]

[6] Consistent selection functions can be considered as interpretations ($\mathbf{I}_f(L)$ is true iff $f(L)$ is selected in a clause). From this point of view hyper tableaux can be seen as a tableau counterpart to *semantic resolution* [57] and would be better called *semantic* tableaux if the latter were not used already for the whole framework.

Theorem 4.6. *If the finite ground clause set S is unsatisfiable, then for any consistent selection function f there is a hyper tableau proof with selection function f for S.*

Proof. The proof is very similar to the proof of Theorem 4.5. Consider a maximal, open branch B in a hyper tableau for S and $\overline{B}_f$. $\overline{B}_f - B$ is consistent, because f is consistent, so when $\overline{B}_f$ is inconsistent there are literals $L \in \{f(C) \mid C \in S, \mathbf{I}_B \not\models C\}$ such that $\overline{L}$ occurs in B. Obtain B' by removing all such literals from $\overline{B}_f$. Now B' is consistent and we claim $\mathbf{I}_{B'} \models S$. By construction $B' \supseteq B$, so it suffices to prove $\mathbf{I}_{B'} \models C$ for clauses C not used on B (this implies $|f(C)| > 0$). This holds, because at least one literal in $f(C)$ is still present in B', otherwise $\overline{f(C)} \subseteq B$ and a hyper extension step is possible with C on B contradicting the assumption that all possible extension steps were made on B. □

4.6.1. Positive Hyper Tableaux

Many hyper tableau calculi focus on one specific selection function f_N which selects exactly the negative literals in a clause. This suggests a notation of clauses as rules of the form

$$p_1, \ldots, p_n \to q_1, \ldots, q_m \tag{4.5}$$

where p_i, q_j are atoms. All literals in the premiss are selected. The ensuing calculi are called *positive hyper tableaux*, because negative literals occur only as leaves of closed branches in a hyper tableau with f_N. Therefore, reduction steps cannot occur and are not required. It is also easy to see that a maximal open branch B has a saturation $B' = B \cup \{\neg p \mid p \in (\mathcal{A} - B)\}$.

4.6.2. First-Order Issues

As before, selection functions should be liftable. On the first-order level for a hyper extension step of branch B with clause C, a set $B_0 \subseteq B$ as well as a simultaneous MGU of $\{\{L, L'\} \mid L \in B_0, L' \in \overline{f(C)}\}$ must be computed. This is easily seen to be an instance of the first-order clause subsumption problem which is NP-hard [20]. This complexity is implicitly present in proof search without hyper steps as well, although on a different level. It certainly does not indicate inferiority of hyper tableaux.

A more interesting question is: how is dealt with completeness problems arising from difficulty of fair substitution selection in destructive calculi? For positive hyper tableaux, several strategies can be found in the literature:

The first option is to fix a heuristics for formula selection and trade incompleteness for success in certain problem domains [13, 54]. Another form of incompleteness (in expressivity) ensues from restriction to range-restricted sets of clauses:

Definition 4.8. *A first-order clause of the form (4.5) is* range-restricted *when all variables occurring in the premiss occur also in the conclusion.*

Observe that positive range-restricted clauses are ground. If S is range-restricted, then a positive hyper tableau for S is ground. As an immediate consequence, fair selection of clauses used in extension steps yields a proof confluent calculus of which efficient implementations were realized [41, 19, 29, 28].

Let us call variables occurring in more than one literal of the conclusion and not in the premiss *critical*. In [4] ground instances of clauses restricted to critical variables are enumerated to obtain a proof confluent calculus which is somewhat better than enumerating all ground instances.

Baumgartner [2] improves on that: like in the range-restricted case, a tableau is treated as if it were ground: substitutions are only applied to instances of input clauses (matching). If an instance $L\sigma$ (with critical variables) of a literal occurrence L on a tableau branch is required, then σ is applied to an instance of the clause containing L and the result is added to the input clause set. The latter possibility regains completeness. Needless to say, lifting is not trivial in such a calculus.

This version of first-order hyper tableaux is not destructive as only input clauses are instantiated. The destructive part of the substitution of the closure rule is recorded "outside" of the tableau as additional instances of input clauses. They can be arranged in a fair manner easily. This can also be seen as a kind of constraint on substitutions to guide the proof search.

In principle, a proof confluent, *destructive* calculus could be obtained if one could compile these "outside" clauses (and their fair selection) into the selection rule. Consequences for such a calculus would be: (i) clause selection is probably not defined branch-local, because "outside" clauses touch on several branches; (ii) one needs to work up to renamability of clauses; (iii) a suitable ordering might be used to enumerate required instances of literals fairly. All three ingredients are actually contained in a very recent suggestion for a proof confluent destructive calculus by Beckert [5].

4.7. Tableaux with Cuts and Lemmas

So far we discussed restrictions of tableau calculi aimed at diminishing the search space. Some problems, however, have extremely long tableau proofs. Already on the ground level there exist classes of formulas S_n such that the smallest tableau proof is exponential in the size of S_n whereas short resolution proofs exist [14]. The reason is that resolution incorporates an analytic cut rule or (and this is just another name) lemma generation.

Example 4.5. Let $\{p_1, \ldots, p_n\}$ be different ground atoms. Consider the clause set

$$S_n = \{L_1 \vee \cdots \vee L_n \mid L_i \in \{p_i, \neg p_i\},\, i \in \{1, \ldots, n\}\} \tag{4.6}$$

Obviously, S_n is unsatisfiable. D'Agostino [15] proved that the smallest closed clause tableau for S_n has at least $n!$ inner nodes, whereas S_n contains merely $n2^n$ literals. Even simple truth table checking has linear cost in the size of S_n.

Now consider the clause set $T_n = \{p_i \vee \neg p_i \mid i \in \{1, \ldots, n\}\}$. $S_n \cup T_n$ has a short proof (displayed for $n = 3$ in Figure 10). The point is that the tautologies in T_n can be used to enumerate all interpretations over $\{p_1, \ldots, p_n\}$. Then each clause in S_n is contradicted by exactly one interpretation. The resulting tableau has $n2^n + 2^{n+1} - 1 \in \mathcal{O}(|S_n \cup T_n|)$ nodes.

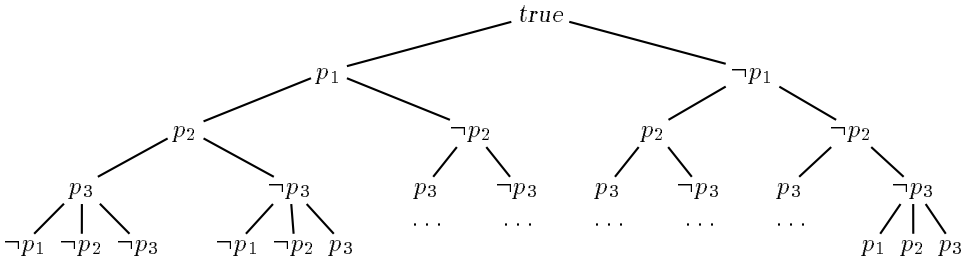


Figure 10. Clause tableau proof for $S_3 \cup T_3$.

The effect of the clauses T_n can also be achieved by adding a new tableau extension rule

$$\frac{}{p \mid \neg p} \tag{4.7}$$

called *atomic cut* rule. It is closely related the well-known cut rule part of sequent calculi [21].

4.7.1. Tableaux and Sequent Calculi

A *propositional clausal sequent* is an expression of the form $\Gamma \Rightarrow \Delta$, where Γ is a tuple of clauses and Δ is a tuple of atoms. $\Gamma \Rightarrow \Delta$ is *valid* iff the formula $\bigwedge_{G \in \Gamma} G \rightarrow \bigvee_{D \in \Delta} D$ is valid. Hence, a clause set S is unsatisfiable if the sequent $S \Rightarrow$ is valid. The propositional clausal sequent calculus consists of only three rules:

$$\frac{\Gamma, L_1, \Gamma' \Rightarrow \Delta \mid \cdots \mid \Gamma, L_n, \Gamma' \Rightarrow \Delta}{\Gamma, L_1 \vee \cdots \vee L_n, \Gamma' \Rightarrow \Delta} \vee\text{-left}$$

$$\frac{\Gamma, \Gamma' \Rightarrow P, \Delta}{\Gamma, \neg P, \Gamma' \Rightarrow \Delta} \neg\text{-left} \qquad \frac{\times}{\Gamma, P, \Gamma' \Rightarrow \Delta, P, \Delta'} \text{axiom} \tag{4.8}$$

It is straightforward to show that a sequent is valid iff it has a proof tree in which all leaves are marked with a $\times$. Moreover, there is a strong duality between clause tableaux and clausal sequent calculi, summarized in Table 3.[7]

[7] This duality extends to the non-clausal and first-order case, see [58].

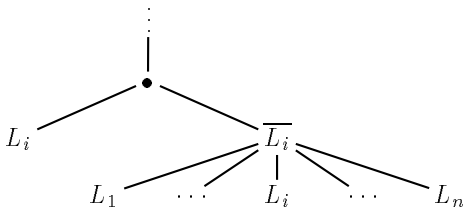
Clausal Sequents	Clause Tableaux
Axiom rule	Closed branch
$\vee$-left rule	Tableau extension
Atoms/unit clauses left of $\Rightarrow$	positive branch literals
Atoms right of $\Rightarrow$	negative branch literals
sequent proofs interpreted as logical conjunction of their end sequents	tableaux interpreted as logical disjunction of their branches
sequents interpreted as logical disjunction of their elements	branches interpreted as logical conjunction of their literals
Validity proof	Unsatisfiability proof

Table 3
Duality between sequent and tableau calculi.

While literal occurrences can be shared among several tableau branches, they are duplicated in sequents. In the light of this and the duality between sequent and tableau proofs, rule (4.7) is exactly the same as the usual cut rule of sequent calculi, where the *cut formula* ϕ is restricted to atomic formulas:

$$\frac{, , , ' \Rightarrow \Delta, \phi, \Delta' \quad \Big| \quad , , \phi, , ' \Rightarrow \Delta, \Delta'}{, , , ' \Rightarrow \Delta, \Delta'} \tag{4.9}$$

4.7.2. Tableaux with Lemmas
The main problem of the atomic cule (4.7) is that its application is completely unrestricted. A first restriction is obtained by allowing its application only if the following extension rule constitutes a strong connection step:



This way no more branches are generated if one had applied the extension step immediately. On the other hand, in $n-1$ of the new branches the literal $\overline{L_i}$ is present. Applying $n-1$ atomic cuts before an extension step and writing the resulting proof

tree as a "macro rule" yields the *extension rule with local lemmas*:

$$\begin{array}{|c|c|c|c|c|}\hline L_1 & \overline{L_1} & \overline{L_1} & \cdots & \overline{\overline{L_1}} \\ & L_2 & \overline{L_2} & \cdots & \overline{\overline{L_2}} \\ & & L_3 & \cdots & \overline{\overline{L_3}} \\ & & & & \vdots \\ & & & \ddots & \overline{L_{n-1}} \\ & & & & L_n \\ \hline \end{array} \qquad L_1 \vee \cdots \vee L_n \in S \tag{4.10}$$

The complemented literals are called *local lemmas*: for example, $\overline{L_i}$ is obtained as a lemma after the current branch B is closed with the help of L_i. (Recall that closed branches are unsatisfiable, so this amounts to $B \models \overline{L_i}$.) The lemma is *local* to certain branches as opposed to being global (on the whole tableau).

4.7.3. Tableaux with Folding Up

Depending on the sequence of branch closures, there are $n!$ different local lemma versions of an extension with an n-literal clause. Not all of these are equally useful. To remedy this situation, a version of local lemma generation called *folding up rule* has been suggested. It can be seen as "lazy lemma generation".

Example 4.6. Consider the clause set $S = \{p \vee t,\ p \vee \neg t,\ \neg p \vee q \vee s,\ \neg q \vee r,\ \neg r \vee \neg p,\ \neg s \vee r\}$ and the partial tableau proof for S displayed in Figure 11. After closure of branch $B = \{p, q, r, \neg p\}$ one knows that $S \cup \{p, q\} \models \neg r$. This lemma is useless, though, because branch $\{p, q, r, \neg r\}$ to the left is closed anyway. But inspection of the proof obtained so far shows that in fact $S \cup \{p\} \models \neg r$ holds, because q was not used to close B.

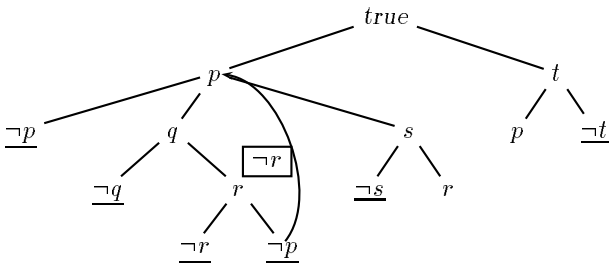


Figure 11. Illustration of Example ex:fold-up.

In *clause tableaux with folding up rule* each local lemma may be moved up in the tableau until the lowest literal that was used in its proof [35, 34] and it can be used to close any branch below its new position.

With the folding up rule, in the previous example lemma $\neg r$ may be moved up to p and lemma $\neg p$ from the right subtree is moved up to *true* (the latter is also possible with a suitable variant of the local lemma rule). In Figure 12 new lemma positions are boxed, moves are indicated by dashed arrows, while additional closures through new lemmas are indicated by solid arrows.

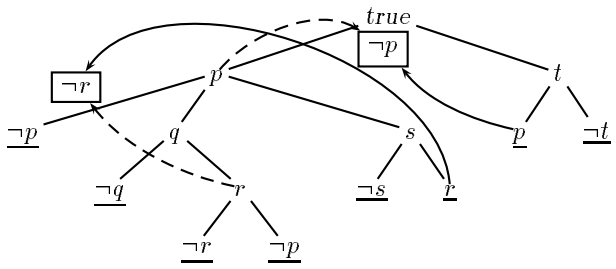


Figure 12. Tableau with folding up rule.

One can show that folding up does not change the *indeterministic power* of tableau with local lemmas that is the length of shortest proofs does not change. In a deterministic implementation folding up can nevertheless be very beneficial, because an ill choice of *select clause* can be remedied.

4.7.4. Tableaux with Factoring

Tableaux with factoring (also called *tableaux with merging*) are closely related to the subsumption rule in resolution theorem proving [52]. Assume branch B is not yet closed, but there is a "more general" branch B' that has been closed already. Formally, a branch B' is *more general* than a branch B iff $B' \subseteq B\sigma$. Instead of trying to close B, one may simply refer to B' then, apply σ to B and consider it as closed. Soundness of tableaux with factoring is straightforward to prove; on the other hand it is sufficient to observe that tableaux with factoring can be simulated by tableaux with local lemmas: assume B was closed by referring to a more general branch B' and that L is the top-most literal on B' not on B. Now replace the extension rule that produced L with a local lemma version putting $\overline{L}$ on B. As B' is more general than B, there must be $L' \in B$ and substitution σ such that $L'\sigma = L$, so B can be closed with σ.

An improvement of tableaux with factoring called *tableaux with regressive merging* was introduced in [63] and is essentially the same as tableaux with the folding up rule; details can be found in [62].

4.7.5. Problems of Strengthening Tableaux

Strengthening of tableau procedures at first seems a sure win, because length of proofs can be drastically reduced. On the propositional level this holds without

reservation and it can safely be claimed that any competitive propositional proof procedure embodies some variant of cut. On the first-order level, however, additional literals introduced as cuts or lemmas create additional possibilities for branch closure. The negative effects from this increase of the search space can easily outweigh the possibility of finding shorter proofs.

Example 4.7. Let $S = \{\neg p(x) \vee q(x) \vee r(x), p(a), \neg q(a), p(b), \neg q(b), \neg r(b)\}$. In the partial connection tableau in Figure 13 the left branch was closed first by extension with $p(a)$ (where only $p(b)$ leads to success). This forces extension with $q(a)$ in the middle branch and generation of some lemmas (framed literals). The lemmas allow to extend the right branch with another instance of the first clause which would have been impossible without them. Detection of the wrong first extension is thus possibly delayed a long time.

In the example, a regularity check would help ($r(a)$ becomes irregular on the rightmost branch) and also restriction of lemma usage to reduction steps. Although this helps somewhat, more complex examples create the same problems as before.

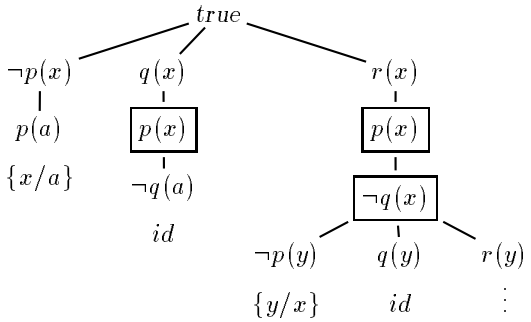


Figure 13. Search space increase caused by local lemmas.

On the other hand, local lemmas are not strong enough on the first-order level. In the clause tableau in Figure 5, for example, the lemma $\neg q(c)$ can be folded up to the *true* node, but is useless to close the open branch on the right, because a different instance is required. But in fact it is justified to derive even $(\forall x)\neg q(x)$ as a lemma. This is always possible when the proof of (that is: the tableau below) the lemma does not instantiate any variables that occur outside of it. It remains to be seen whether such an optimization can be efficiently implemented and does not blow up the search space beyond any usefulness.

4.8. Tableaux and Logic Programming

Some calculi for extensions of logic programs have a natural interpretation as variants of clause tableaux. This includes [47, 48, 40, 51, 3]. Their treatment is beyond

the scope of this course.

5. Historical Remarks

Despite their recent flourishing, the history of tableau methods is much older than that of resolution. They can be traced back to the cut-free version of Gentzen's sequent calculus [21]. Hintikka [30] and Beth [10] abstracted from structural rules in sequent calculi (essentially treating sequents as sets of formulas), improved the proof representation, and introduced signed formulas. They also stressed the semantic view of tableaux as a procedure that tries systematically to find a counter example for a given formula (a model in which its negation is true) as opposed to Gentzen's purely proof theoretical motivation.[8] Further improvements were made by Schütte [53]; Smullyan's elegant formulation [58], employing unifying notation which greatly simplified matters, became very popular while similar contributions by Lis [37] unfortunately went unnoticed.

Many important improvements directed to automated proof search in sequent/tableau/model elimination calculi were made quite early: free variables [31, 50], unification [38, 11, 1], proof representation and connection refinements [17, 11, 1].

The relative success of resolution-based theorem proving, however, eclipsed this progress and serious implementations of tableau-like calculi are spurious before the late 1980s [13, 44].

In the last decade tableau-like calculi became focal points of research again, spurred by the success of efficient implementations and the demand for a computational treatment of non-classical logics. The tableau community gathers in an international conference,[9] where many of the relevant results are now published. The Handbook of Tableau Methods [16] contains extensive and fairly up-to-date information, not only with respect to automated reasoning.

References

[1] Peter B. Andrews. Theorem proving through general matings. *JACM*, 28:193–214, 1981.

[2] Peter Baumgartner. Hyper Tableaux — The Next Generation. In Harrie de Swart, editor, *Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Oosterwijk, The Netherlands*, number 1397 in LNCS, pages 60–76. Springer-Verlag, 1998.

[3] Peter Baumgartner and Uli Furbach. Model Elimination without Contrapositives and its Application to PTTP. *Journal of Automated Reasoning*, 13:339–359, 1994.

[4] Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper tableaux. Technical Report 8/96, Institute for Computer Science, University of Koblenz, Germany, 1996. http://www.uni-koblenz.de/universitaet/fb4/publications/GelbeReihe/RR-8-96.ps.gz.

[8] The proof theoretic tradition is very much alive today as witnessed, for example, by Girard's work [22], but it is not really relevant for automated deduction.

[9] `http://www.cs.albany.edu/ nvm/tab99/`

[5] Bernhard Beckert. *Integrating and Unifying Methods of Tableau-based Theorem Proving*. PhD thesis, University of Karlsruhe, Department of Computer Science, July 1998.

[6] Bernhard Beckert and Rajeev Goré. Free variable tableaux for propositional modal logics. In *Proc. International Conference on Theorem Proving with Analytic Tableaux and Related Methods, Pont-a-Mousson, France*, volume 1227 of *LNCS*, pages 91–106. Springer-Verlag, 1997.

[7] Bernhard Beckert and Reiner Hähnle. Analytic tableaux. In W. Bibel and P. Schmitt, editors, *Automated Deduction: A Basis for Applications*, volume I, chapter 1. Kluwer, 1998. To appear.

[8] Bernhard Beckert and Joachim Posegga. leanTAP: Lean tableau-based deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995.

[9] Karel Berka and Lothar Kreiser, editors. *Logik-Texte. Kommentierte Auswahl zur Geschichte der modernen Logik*. Akademie-Verlag, Berlin, 1986.

[10] Evert W. Beth. Semantic entailment and formal derivability. *Mededelingen van de Koninklijke Nederlandse Akademie van Wetenschappen, Afdeling Letterkunde, N.R.*, 18(13):309–342, 1955. Partially reprinted in [9].

[11] Wolfgang Bibel. Tautology testing with a generalized matrix method. *Theoretical Computer Science*, 8:31–44, 1979.

[12] Wolfgang Bibel. *Automated Theorem Proving*. Vieweg, Braunschweig, second revised edition, 1987.

[13] Frank Malloy Brown. Towards the automation of set theory and its logic. *Artificial Intelligence*, 10(3):281–316, 1978.

[14] Stephen Cook and Robert Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36ff, 1979.

[15] Marcello D'Agostino. Are tableaux an improvement on truth tables? Cut-free proofs and bivalence. *Journal of Logic, Language and Information*, 1:235–252, 1992.

[16] Marcello D'Agostino, Dov Gabbay, Reiner Hähnle, and Joachim Posegga, editors. *Handbook of Tableau Methods*. Kluwer, Dordrecht, to appear in 1998.

[17] Gennady V. Davydov. Synthesis of the resolution method with the inverse method. *Journal of Soviet Mathematics*, 1:12–18, 1973. Translated from Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Mathematicheskogo Instituta im. V. A. Steklova Akademii Nauk SSSR, vol. 20, pp. 24–35, 1971.

[18] Melvin C. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, New York, second edition, 1996.

[19] Hiroshi Fujita and Ryuzo Hasegawa. A model generation theorem prover in KL1 using a ramified-stack algorithm. In Koichi Furukawa, editor, *Proceedings 8th International Conference on Logic Programming, Paris/France*, pages 535–548. MIT Press, 1991.

[20] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, San Francisco, 1979.

[21] Gerhard Gentzen. Untersuchungen über das Logische Schliessen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English translation [61].

[22] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[23] Christian Goller, Reinhold Letz, Klaus Mayr, and Johann Schumann. SETHEO V3.2: recent developments. In Alan Bundy, editor, *Proc. 12th Conference on Automated Deduction CADE, Nancy/France*, LNAI 814, pages 778–782. Springer-Verlag, 1994.

[24] Reiner Hähnle and Gonzalo Escalada-Imaz. Deduction in many-valued logics: a survey. *Mathware & Soft Computing*, IV(2):69–97, 1997.

[25] Reiner Hähnle and Stefan Klingenbeck. A-ordered tableaux. *Journal of Logic and Computation*, 6(6):819–834, 1996.

[26] Reiner Hähnle, Neil Murray, and Erik Rosenthal. Completeness for linear regular negation normal form inference systems. In Zbigniew W. Raś and Andrzej Skowron, editors, *Foundations of Intelligent Systems, 10th International Symposium, ISMIS'97, Charlotte, North Carolina, USA*, number 1325 in LNCS, pages 590–599. Springer-Verlag, 1997.

[27] Reiner Hähnle and Christian Pape. Ordered tableaux: Extensions and applications. In Didier

Galmiche, editor, *Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Pont-à-Mousson, France*, volume 1227 of *LNCS*, pages 173–187. Springer-Verlag, 1997.

[28] Ryuzo Hasegawa, Hiroshi Fujita, and Miyuki Koshimura. MGTP: a model generation theorem prover—its advanced features and applications. In Didier Galmiche, editor, *Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Pont-à-Mousson, France*, volume 1227 of *LNCS*, pages 1–15. Springer-Verlag, 1997.

[29] Ryuzo Hasegawa, Miyuki Koshimura, and Hiroshi Fujita. MGTP: A parallel theorem prover based on lazy model generation. In Deepak Kapur, editor, *Proc. 11^{th} International Conference on Automated Deduction*, LNAI 607, pages 776–780. Springer-Verlag, 1992.

[30] K. J. J. Hintikka. Form and content in quantification theory. *Acta Philosohica Fennica*, 8:7–55, 1955.

[31] Stig Kanger. *Provability in Logic*, volume 1 of *Acta Universitatis Stockholmiensis*. Almqvist & Wiksell, Stockholm, 1957.

[32] Stefan Klingenbeck and Reiner Hähnle. Semantic tableaux with ordering restrictions. In Alan Bundy, editor, *Proc. 12th Conference on Automated Deduction CADE, Nancy/France*, volume 814 of *LNCS*, pages 708–722. Springer-Verlag, 1994.

[33] R. E. Korf. Depth-first iterative deepening: an optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[34] R. Letz, K. Mayr, and C. Goller. Controlled integration of the cut rule into connection tableau calculi. *Journal of Automated Reasoning,*, 13(3):297–338, December 1994.

[35] Reinhold Letz. *First-Order Calculi and Proof Procedures for Automated Deduction*. PhD thesis, TH Darmstadt, June 1993.

[36] Reinhold Letz, Johann Schumann, Stephan Bayerl, and Wolfgang Bibel. SETHEO: A high-perfomance theorem prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.

[37] Z. Lis. Wynikanie semantyczne a wynikanie formalne (logical consequence, semantic and formal. *Studia Logica*, 10:39–60, 1960. Polish, with Russian and English abstracts.

[38] Donald W. Loveland. A simplified format for the model elimination procedure. *Journal of the ACM*, 16(3):233–248, July 1969.

[39] Donald W. Loveland. *Automated Theorem Proving. A Logical Basis*, volume 6 of *Fundamental Studies in Computer Science*. North-Holland, 1978.

[40] Donald W. Loveland. Near-Horn Prolog and beyond. *Journal of Automated Reasoning*, 7:1–26, 1991.

[41] Rainer Manthey and François Bry. SATCHMO: A theorem prover implemented in Prolog. In *Proceedings 9th Conference on Automated Deduction*, LNCS, New York, pages 415–434. Springer-Verlag, 1988.

[42] Max Moser, Ortrun Ibens, Reinhold Letz, Joachim Steinbach, Christoph Goller, Johann Schumann, and Klaus Mayr. SETHEO and E-SETHEO—the CADE-13 systems. *Journal of Automated Reasoning*, 18(2):237–246, April 1997.

[43] Andreas Nonnengart, Georg Rock, and Christoph Weidenbach. On generating small clause normal forms. In Hélène Kirchner and Claude Kirchner, editors, *Proc. 15th International Conference on Automated Deduction, Lindau, Germany*, number 1421 in LNCS, pages 397–411. Springer-Verlag, 1998.

[44] F. Oppacher and E. Suen. Controlling deduction with proof condensation and heuristics. In Jörg H. Siekmann, editor, *Proc. 8th International Conference on Automated Deduction*, pages 384–393, 1986.

[45] Christian Pape. Vergleich und Analyse von Ordnungseinschränkungen für freie Variablen Tableau. (In German). Interner Bericht 30/96, Universität Karlsruhe, Fakultät für Informatik, 1996.

[46] Christian Pape and Reiner Hähnle. Restart tableaux with selection function. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Fifth Kurt-Gödel-Colloquium, KGC'97, Vienna*, volume 1289 of *LNCS*, pages 219–232. Springer-Verlag, 1997.

[47] David A. Plaisted. Non-Horn clause logic programming without contrapositives. *Journal of*

Automated Reasoning, 4:287–325, 1988.

[48] David A. Plaisted. A sequent-style model elimination strategy and a positive refinement. *Journal of Automated Reasoning*, 6:389–402, 1990.

[49] David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[50] Dag Prawitz. An improved proof procedure. *Theoria*, 26:102–139, 1960. Reprinted in [55].

[51] David W. Reed and Donald W. Loveland. A comparison of three prolog extensions. *Journal of Logic Programming*, 12:25–50, 1992.

[52] J. A. Robinson. A machine-oriented logic based on the resolution principle. *JACM*, 12(1):23–41, January 1965. Reprinted in [55].

[53] Kurt Schütte. *Beweistheorie*, volume 103 of *Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen mit besonderer Berücksichtigung der Anwendungsgebiete.* Springer-Verlag, 1960.

[54] Benjamin Shults. A framework for using knowledge in tableau proofs. In Didier Galmiche, editor, *Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Pont-à-Mousson, France*, volume 1227 of *LNCS*, pages 328–342. Springer-Verlag, 1997.

[55] Jörg Siekmann and Graham Wrightson, editors. *Automation of Reasoning: Classical Papers in Computational Logic 1957–1966*, volume 1. Springer-Verlag, 1983.

[56] Jörg Siekmann and Graham Wrightson, editors. *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 2. Springer-Verlag, 1983.

[57] James R. Slagle. Automatic theorem proving with renamable and semantic resolution. *Journal of the ACM*, 14(4):687–697, 1967. Reprinted in [56].

[58] Raymond M. Smullyan. *First-Order Logic.* Dover Publications, New York, second corrected edition, 1995. First published 1968 by Springer-Verlag.

[59] Mark E. Stickel. A Prolog technology theorem prover: A new exposition and implementation in Prolog. *Theoretical Computer Science*, 104(1):109–129, 1992.

[60] Geoff Sutcliffe and Christian Suttner. The results—of the CADE-13 ATP system competition. *Journal of Automated Reasoning*, 18(2):271–286, April 1997.

[61] M. E. Szabo, editor. *The Collected Papers of Gerhard Gentzen.* North-Holland, Amsterdam, 1969.

[62] Kevin Wallace. *Proof Truncation Techniques in Model Elimination Tableaux.* PhD thesis, University of Newcastle, Australia, December 1994.

[63] Kevin Wallace and Graham Wrightson. Regressive merging in model elimination tableau-based theorem provers. *Journal of the Interest Group in Pure and Applied Logics*, 3(6):921–938, October 1995. Special Issue: Selected Papers from Tableaux'94.